Constellation: A Science Graph Network for Scalable Data and Knowledge Discovery in Extreme-Scale Scientific Collaborations

Sudharshan S. Vazhkudai, John Harney, Raghul Gunasekaran, Dale Stansberry, Seung-Hwan Lim, Tom Barron, Andrew Nash and Arvind Ramanathan
Oak Ridge National Laboratory
{*vazhkudaiss, harneyjf, gunasekaranr, stansberrydv, lims1, tbarron, nashaw, ramanathana*}*@ornl.gov*

***Abstract*—Constellation's overarching goal is the federation of information from resources within an extreme-scale scientific collaboration to enable the scalable discovery of data and new knowledge pathways. The resource fabric is comprised of petascale supercomputers and storage systems, users, jobs, datasets and lifecycle artifacts. For an extreme-scale supercomputing center, normal operations can generate hundreds of millions of data products and metadata entries describing the resource fabric. Constellation federates the information extracted from the resources using a custom, transformative science graph network; constructs rich metadata indexes and higher-order derived metadata from the extracted information; and conducts scalable graph analytics to unravel hidden data pathways. Our implementation and deployment for a production, supercomputing facility shows that the graph can scale to more than 750 million vertices, its domain agnostic indexing can answer interesting science queries, and its analytics can aid in structural, topological and temporal analysis to identify usage hotspots.**

I. Introduction

Extreme-scale simulations of complex physical phenomena on leadership-class supercomputers, e.g., Titan [4], Mira [17] and other machines on the Top500 [10] list, produce tens of millions of data products that need to be discovered, correlated and analyzed by a distributed community to glean insights. Titan, for example, is the U.S. Department of Energy's (DOE) 27-petaflop machine at the Oak Ridge Leadership Computing Facility (OLCF). It routinely runs massively parallel simulations from several grand challenge application problems in science domains such as combustion, fusion, astrophysics, materials, molecular dynamics and climate. For example, a single 24-hour, 256,000-core simulation run of the Fusion application, XGC [5], on Titan (No. 3 on the Top500 list) produces 1 PB of data, spread across O(100,000) files, each time step (typically every hour). A large-scale simulation is typically followed by post-processing or data analyses to reduce and analyze the simulation data products. Oftentimes, the simulation and data analyses are run by different user groups, frequently at different times. In the XGC example above, post-processing results in an order of magnitude data reduction from 1 PB to 100 TB, and may be conducted either immediately after the simulation or weeks, even months later. The simulation and data analysis jobs are often part of a larger, yearlong "campaign" comprised of hundreds of such "hero" runs. Such a collaboration requires the capture of metadata surrounding data production and associated processes, leading up to the publication and curation of artifacts, to facilitate scalable data discovery.

Constructing collaborative software for science has always been a daunting challenge. A key weakness herein is the inability to sufficiently capture the complex interrelationships among the participating entities, resulting in isolation gaps. The diverse nature of the resources within a collaboration, e.g., data products, processes, users, publications, curation artifacts, clusters and file systems, and their spatial and temporal connections make it a significant challenge for extant systems to capture these relationships. In such an environment, where resource and connectivity information is so scarce, users themselves are many times unable to make sense of their prior job runs and experiment setup. This difficulty is compounded for collaborations or researchers interested in the problem at a later date. Current state-of-the-art still depends on users manually providing metadata, an extremely human-intensive and error-prone process. In this setting, data discovery is extremely cumbersome or impossible.

To this end, we propose two fundamentally novel concepts. The first investigation delves into the creation of a transformative, *science graph relationship network* structure that bridges the isolation gaps within a collaboration. The science graph network serves as a scalable way to both *federate* and *correlate* information (metadata) from the resource fabric of the collaboration. We posit that viewing a scientific collaboration through the prism of a graph connectivity network allows us to build complex associations among resources, and discover new data pathways by both exploiting graph properties and performing graph data analytics. We argue that new knowledge pathways can be discovered via the graph network approach that would normally be impossible with the current state-of-the-art.

To build the science graph, we need rich metadata about the resources in the collaboration. This leads us to the second concept, which is a foray into the construction of rich *knowledge indexes* atop information extracted from the resource fabric. The multitudes of resources, e.g., data, jobs, publications, and the systems that host them have a wealth of metadata buried within, which if harnessed efficiently, can help with numerous data disposition questions, without requiring human intervention. Therefore, a key challenge we address is the non-intrusive construction of sophisticated knowledge structures

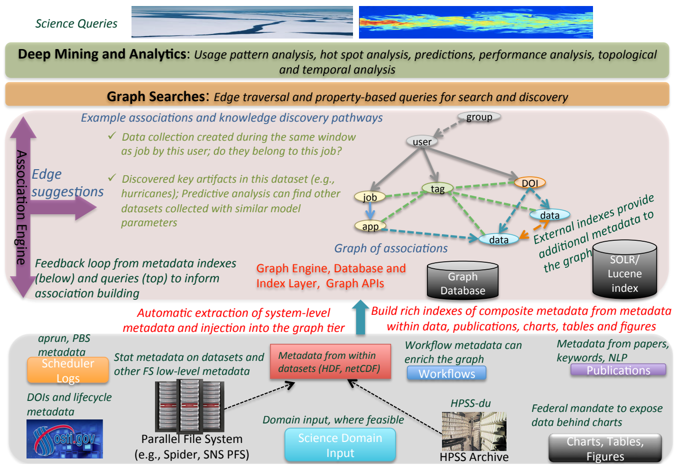
Science Queries
Deep Mining and Analytics: Usage pattern analysis, hot spot analysis, predictions, performance analysis, topological and temporal analysis
Graph Searches: Edge traversal and property-based queries for search and discovery
Example associations and knowledge discovery pathways
✓ Data collection created during the same window as job by this user; do they belong to this job?
✓ Discovered key artifacts in this dataset (e.g., hurricanes); Predictive analysis can find other datasets collected with similar model parameters
Association Engine
Edge suggestions
group
user
job
tag
DOI
app
data
data
Graph of associations
External indexes provide additional metadata to the graph
Feedback loop from metadata indexes (below) and queries (top) to inform association building
Graph Engine, Database and Index Layer, Graph APIs
Graph Database
SOLR/ Lucene index
Automatic extraction of system-level metadata and injection into the graph tier
Build rich indexes of composite metadata from metadata within data, publications, charts, tables and figures
aprun, PBS metadata
Scheduler Logs
Stat metadata on datasets and other FS low-level metadata
Metadata from within datasets (HDF, netCDF)
Workflow metadata can enrich the graph
Workflows
Metadata from papers, keywords, NLP
Publications
DOIs and lifecycle metadata
osti.gov
Parallel File System (e.g., Spider, SNS PFS)
Domain input, where feasible
Science Domain Input
HPSS-du
HPSS Archive
Federal mandate to expose data behind charts
Charts, Tables, Figures

Fig. 1: Constellation Overview. The figure shows our bottom-up approach of extracting metadata from the resource fabric, and federating them into a scalable science graph network infrastructure that facilitates the discovery of new knowledge/data pathways via rich data mining, analytics and graph querying.

with minimal user or domain input.

Contributions: We propose the *Constellation* science graph network system with the following contributions. (i) We describe resources using automatic metadata extraction and refinement strategies, building multi-layer metadata index structures and derived metadata from the base metadata. (ii) We use the metadata and indexes to build a science graph network. We build rich graph associations and search indexes to correlate and discover the resources within the collaboration. (iii) We explore how graph analytics can be layered atop the basic graph to discover new data pathways. Together, these concepts will bring about a profound impact on knowledge discovery in scientific collaborations. Our design, implementation and analysis demonstrates the viability of the research ideas put forth, and showcases the challenges involved in constructing a solution for extreme-scale HPC centers with over a billion graph vertices and associated metadata.

II. Constellation Design and Implementation

We create a transformative "science graph relationship network" that federates information from resources such as systems, users, data and processes in a collaboration to enable the scalable discovery of data pathways (Figure 1). To this end, we focus on the following key challenges: (i) federating and connecting the resources in a collaboration using a graph construct to alleviate the isolation gaps, (ii) enabling the automatic capture of information regarding the diverse resources and describing them with rich metadata structures, and (iii) building associations through intelligent correlation of information sources, and enabling search and analytics on the graph to discover new data products.

Science Graph Network: Inspired by the linked data initiatives of the Web, we propose to view a collaboration as a sophisticated "science graph network" that federates information from diverse resources and captures their interrelationships to enable scalable data discovery. The graph is meant to act as an easily traversed web of metadata vertices, data, information, and knowledge resources, providing both humans and machines the ability to reason about them. We posit that the graph network can answer intuitive queries and scientific questions about the collaboration and its products that would normally be impossible or inefficient to process using the current state-of-the-art. The graph structure has been widely deployed in social networking to derive relationships between users and products. Imagine a similar concept extrapolated to a science network, wherein the graph is used to automatically and intelligently *build associations among resources*, e.g., new data products of interest to a user based on his own scientific conduct. Sections II-A and II-C explore this theme.

Knowledge Representation: The construction of this graph is highly dependent upon the improvement of the current state-of-the-art in resource information collection methodologies. More specifically, it is crucial that we improve the knowledge representation of collections of data products and artifacts in many areas. We propose a bottom-up approach of automatically extracting metadata from the resource fabric, collecting as much information as possible without user intervention. We describe data as thoroughly as possible by using flexible and scalable high performance indexes. We derive as much metadata as we can from base metadata extracted from resources, by designing a multi-layered metadata index structure capable of supporting ex post facto analytics on base metadata and properties. Sections II-B and II-D discuss this theme.

Graph analytics and mining: Beyond the graph associations, deeper levels of knowledge discovery require analytical techniques and mining. We will explore the structural, topological and temporal dimensions within the science graph to glean deeper insights into the usage, complex association network and hidden knowledge pathways. We discuss scalable analytics on the graph that can provide novel tools to discover relevant data products. Section II-E explores this theme.

Design Criteria: Any graph based implementation must address several key concerns. Wei et al. [25] gave a comprehensive outline of the challenges by describing six essential criteria for building a satisfactory graph implementation in a large-scale environment. The criteria are: (1) strong scalability to adapt to the ever increasing amount of data, (2) high space efficiency to be contained in RAM for high access efficiency, (3) high query performance to resist the rising search complexity during the scaling-up process, (4) controllable query accuracy to resist errors introduced by compact data structures, (5) element location capability to inform the system where to locate the possible elements, and (6) element deletion capability to allow the storage system to recycle space. We validated our design by addressing these criteria.

Criterion (1) pertains to the problems at an HPC facility. We address this by building tools for extracting information from the resource fabric. This is addressed in Section II-B. Criterion (2) pertains to the time and space considerations in graph construction. We address this with an in-memory graph structure that differs in design from traditional offerings by trading certain features for space efficiency. This is described in Section II-A. Criterion (3) presents the issue of searching across high volume content offered by an HPC facility. We address this by adopting a decentralized index design capable of decoupling and grouping different types of metadata

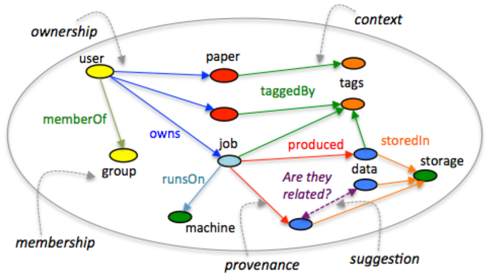

Fig. 2: Resource entities and their relationships in the graph. The figure shows different types of resources such as users, jobs, data, and tags. It further shows several types of edges or relationships by which the resources are connected.

streams into individual high-performance and scalable indexes, discussed in Section II-D. Criterion (4) deals with query recall accuracy. We address this by using a novel hierarchical index abstraction (discussed in Section II-D3) capable of deriving new metadata for data products. The new metadata provides richer descriptions of the material, improving the recall rates and enhances discovery of new knowledge pathways. Criterion (5) refers to the location of events of interest within the collaboration. These events include graph hotspots, popularity, and temporal evolution of facility resource usage. We address this by adding an analytics layer (Section II-E) to perform topological and structural analysis for deeper insights. Criterion (6) refers to the ability to adapt to a dynamic and ever-changing environment typical of any storage system in a petascale facility. We address this by building a highly efficient difference engine (described in Section II-B that compares daily snapshots of the state of the facility.

A. Graph

Constellation's base structure is a *graph network* with the diverse resources of the collaboration represented as vertices or nodes in the graph. Resource types include the following:

- *organizational* resources, comprised of system users and allocation projects to which they have been assigned,
- *process* resources, comprised of large-scale application simulation and data analysis jobs that run on the supercomputers and analysis clusters,
- *data* resources, comprised of output files from application simulation and analysis jobs, which reside on the HPC center's filesystems and archival storage, and
- *conceptual* resources, comprised of user specified resource tags and curated products like digital object identifiers (DOIs).

Given the resources, we need to identify and document the relationships among them. In general, graph databases are well-suited for supporting systems requiring complex, flexible, or evolving data schemas, such as scientific collaborations. Because relationship information (represented by graph edges) is local to, or indexed from, resources (represented by graph vertices) within the graph, graph databases are especially suited to satisfying complex queries over large sets of resources. Such a graph infrastructure can elegantly capture the many to one and many to many relationships between and among these entities. For example, jobs and their data products can be linked to the facility users that initiated them.

TABLE I: Constellation Graph size

Data Source at OLCF	Vertex Type	Count
Lester (Spider parallel file system snapshot)	file	750 Million
HPSSDU (HPSS tape archive snapshot)	file	61 Million
RATS (accounting)	user	13K
RATS	group	18K
scheduler logs on Titan	jobs	500/day
scheduler logs on Titan	apps	8K/day

The graph structure offers a natural, non-hierarchical view into the interactions among collaboration resources, alleviating connectivity and isolation gaps.

Constellation represents relationships among resources as edges, as shown in Figure 2. Relationships form strong, but flexible organizational structures and answer questions about relevance and dependency among the resources. The graph allows relationships of any type to be created at runtime between any two vertices. Since in a real-world scientific collaboration, resources can share complex and diverse relationships, we explored a broad set of semantics for edges, namely *Ownership*, e.g., user owns a job; *Membership*, e.g., user belongs to a project or a group; *Component*, e.g., these files belong to this directory or the apps are part of this job; *Provenance* captures the lineage of the resources, e.g., this job produced these datasets; *Context* connects a resource to a new, larger structure e.g., data or job is part of a DOI or tag; and *Suggestion* indicates a prediction of a related data product (or any resource) via the correlation of metadata sources or analytics on the graph. The key challenge here is identifying and asserting relationships from available resource metadata.

While most open source solutions scale by caching portions of the graph on disk, Constellation requires that all of the graph be kept in memory to facilitate the complex queries it is designed to support. To achieve a reduced memory footprint, we used node properties only to capture file ownership. While node properties, which are defined as attributes associated with entities in the graph, allow for very fast queries on the graph, they also consume significant space in memory. Thus we traded lookup speed for memory space to make it possible to keep the whole graph in memory.

Most of the resources in Constellation represent files residing on the parallel file system. Extracting this metadata is a costly operation and can only be done once per day and all updates must be loaded to Constellation at once. Our customized solution facilitates this bulk update in ways currently available graph systems cannot.

B. Information Sources for Graph Vertices

Now that we have defined the resource types that can be represented by vertices in the graph, we will describe how that information is extracted from the systems that host the underlying resources (users, groups, datasets, and jobs) using the OLCF HPC center as an example. While the details of the tools may vary based on site-specific constraints, our goal here is to illustrate some of the common challenges seen in extracting metadata from widely-used systems such as accounting databases, Torque/PBS scheduler logs, the Lustre-based file system and the HPSS archive. Since these systems support production HPC operations, we must also extract information without imposing significant overhead. The two challenges to gathering information from such diverse systems are that they are neither designed to a) interface externally nor b) transform the extracted information into a format suitable for ingestion into the Constellation graph. As a result, we include mechanisms for extracting and reformatting the information without adding overhead.

Table I shows the resource type counts in Constellation for the OLCF case study. All of the organizational resources, e.g., users and group information, are captured in the *Resource Allocation and Tracking System (RATS)* database. We developed a tool that periodically queries this database and updates the Constellation graph accordingly. Similarly, we extract users' jobs from Titan's scheduler logs. This metadata is obtained by parsing two different log streams from Torque [20] (a queue manager) and PBS [12] (a batch system).

We obtain file information via file metadata from the Lustre-based Spider storage system, which currently stores over 750 million files and directories (Table I). Tracking a constantly changing 750 million entry file system can be very arduous. Using *"ls -l"* on such a large-scale file system will simply not scale. Alternatively, if the file system has the *change log* feature enabled, we could use that to capture all changes and populate the graph. However, due to the potentially large log sizes (tens or hundreds of GBs), OLCF had not enabled change logs on Spider. Therefore, our solution is to capture a daily snapshot of the file system metadata and identify the changes on the file system between consecutive snapshots. HPC centers often create file system snapshots to monitor space usage by user and project (e.g., *Lester* tool developed by OLCF). We leverage this information to produce a daily delta of file system changes. The challenge then is to perform a *diff* between two snapshot files with over 750 million entries. We developed a customized difference engine (using an Apache Spark over Hadoop solution [21]), to compute the difference as files are created, updated or deleted on the file system. The result is then applied to the Constellation graph to produce an up-to-date set of file vertices. The full end-to-end process of this engine takes considerable time (hours), so we deploy it as a background process once per day.

For the archival storage system, HPSS, we developed a tool HPSSDU, which produces a snapshot of the metadata. Run once daily, HPSSDU reads the DB2 database where HPSS stores its metadata to extract and format information about each file stored in the archive. The information retrieved corresponds to the normal POSIX inode information, like that retrieved from Lustre: pathname, owner id, group id, creation time, access permissions, etc.

In the above discussion, we have highlighted the challenges of extracting information from extreme-scale file systems and archival storage. The process of creating snapshots for the parallel file system and the archive consumes several hours and is exacerbated by the explosive growth in storage (Table I). At future exascale volumes, it is estimated that at current rates, the daily processing will exceed 24 hours. The above challenges exemplify the difficulties of creating a system displaying near real time information.

C. Associations

The association engine (Figure 1) within the science graph network is responsible for building the connectivity between the nodes (resources). We categorize the associations into the following: (i) basic connectivity, (ii) complex associations and (iii) user-specified views and context. Such rich associations are needed to answer sophisticated user queries.

1) Basic Connectivity: The science graph will need to use the federated resource information to automatically and intelligently build associations between the resources. The base graph is a set of vertices and *basic connectivity* that does not require complex correlation of resource information. For example, straightforward connectivity might include links such as users' *membership* to a group or a project. These edges represent *facts* in the graph that can be derived using standard system commands. The association engine is responsible for asserting such facts in the graph. The facts themselves can help answer several interesting queries, e.g., "show all the DOIs belonging to the biophysics project," while also laying the foundation for more advanced search queries (Section II-D).

2) Complex Associations: Once we have a graph with base-level connectivity, we then derive more speculative relationships between resources based on correlating the wealth of metadata sources. The goal here is to see how we can build associations automatically, based on metadata extracted from resources and derived indexes, without requiring user intervention. To illustrate this type of association building further, consider the *Facebook example of finding friends based on overlapping stints at an institution* [6]. The equivalence here is finding if a data collection belongs to a job based on overlapping times. In the absence of a user or a workflow linking jobs and data products together, how can the graph association engine automatically deduce this relationship? We derive such relationships automatically by correlating metadata sources, e.g., job scheduler logs and file system stat information. For a given user's job, we can obtain start and end times from the scheduler log; based on this job window, the association engine can query the file system stat metadata for data products created by the user during the same time window. Thus, there is a *likelihood* that the data products may be related to the job in question, but it is not certain as the datasets could belong to another job that ran at the same time. The association engine marks this potential relationship discovery between the job and the data collection with a *"suggestion edge,"* indicating the uncertainty involved. It can further assign weights or even a probability to denote the level of confidence in the association, e.g., in addition to the scheduler logs, correlating the job script metadata can result in some keywords corresponding to the datasets (directory or file names), prompting the association engine to give the edge a higher probability. The graph association engine makes such connections between resources in an attempt to discover new data pathways. Extant data discovery and workflow systems (e.g., Earth System Grid Federation [7], Pegasus [1]) are simply unable to capture such advanced relationships, missing a huge opportunity for discovering new knowledge pathways. In Section II-E, we will delve into more sophisticated graph analytics techniques to derive connections between related data products.

3) User-specified Views and Context: Consider a desktop user marking several of his pictures with an index term or a *"tag"* for quick retrieval. Imagine a similar idea extrapolated to the science graph, wherein users can collaboratively add a richer context to the graph by adding their own tags to associate subsets of resources in order to quickly identify and operate them. Tags can offer users a *personalized view* of the science graph, and can be shared with other users. For example, a group of astrophysics users working on a paper submission to the Supercomputing conference may wish to create a tag, *"Supernova-SC16"* that associates their jobs on supernova explosions, data and collaborators so that the resources can be quickly retrieved and operated on. The tag name, the associations and the user-provided metadata become part of the graph, and will be available for search and discovery. In the future, another user can pose a query such as "show me all SC papers on supernovae explosions that used the Titan supercomputer." Our current implementation of the graph can support the creation of such "conceptual vertices."

Similar to tags, digital object identifiers (DOI) is another resource that we can create in the science graph, enabling the association of lifecycle/curation metadata and resource artifacts (e.g., jobs, data and publications). While not currently implemented, DOI resources can also help the graph association engine with inferring automatic associations. If a user used a specific dataset for discovering key artifacts (like hurricanes) and documented it via a DOI in the graph, it gives the graph a reference dataset (and associated resources like publications) for that artifact to discover related data products. The graph connectivity and association engine allows us to build and explore such richer concepts.

D. Hybrid Indexes for Improved Search

Ultimately, the most important feature of Constellation is its ability to allow users to search, discover and disseminate resources of the scientific collaboration. In this vein, the harvesting and cataloging of *metadata* are necessary operations that must be undertaken to support search. An intuitive first step is to leverage the construction of the Constellation graph to enrich our metadata knowledge. We can abstract a graph operation as a computational task to determine whether graph G contains graph pattern H from a user. Abstractly, this can be seen as the *NP-complete* problem known as subgraph isomorphism. Thus, if we completely rely on graph traversals to find H, the operations will be computationally intractable as G grows. Unfortunately, as we are building a customized solution because of the limitations imposed by the HPC environment (see Section II-A), we are not able to leverage the powerful property graph features that are prominent in open-source and commercial graph engines. Instead, our solution employs a decentralized, *hybrid* approach for harvesting and storing descriptive properties of the various resources in the graph. This approach decouples the vast metadata stores with the graph, so that each component focuses on one particular aspect of description, utilizing both the flexibility of navigating through relationships in the graph as well as leveraging high-performance search indexes that are *external* to the graph.

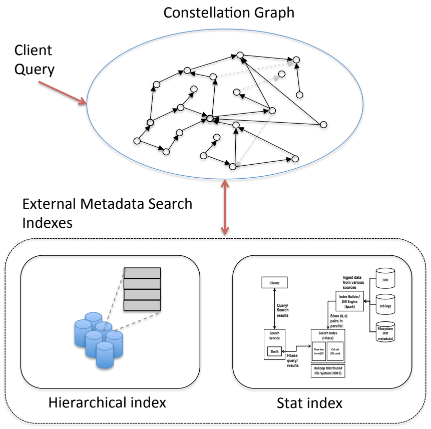
Constellation Graph
Client Query
External Metadata Search Indexes
Clients
Query/ Search results
Ingest data from various sources
Index Builder/ Diff Engine (Spark)
DOI
Job logs
Filesystem stat metadatat
Store (k,v) pairs in parallel.
Search Service
Thrift
Hbase query/ results
Search Index (Hbase)
Row key term|ID
Col val UID, rank
Hadoop Distributed File System (HDFS)
Hierarchical index
Stat index

Fig. 3: Overview of decentralized approach for storage and discovery of metadata. A client issues a query to Constellation, which dispatches it to the two external metadata indexes (Hierarchical Index and Stat Index). It collates the results from the external indexes and combines graph traversals to satisfy complex queries.

To enable property-based queries in this environment, we catalog metadata in separate stores. Specifically, we distribute these properties amongst three different data structures. The first is within the graph itself, where relationships are naturally defined through the edges in a variety of ways (see Section II-A for relationship types) and resources are defined by a universal resource node ID. The second component catalogs the properties we define as system, or *stat* metadata, which include simple key-value properties that represent basic characteristics of the resources such as timestamps, names, and access properties. This information is held in an inverted index external to the graph using the Spark framework [21] and references node IDs in the graph. The final component is also held externally in a *hierarchical* inverted index data structure using the open source Solr framework [2]. While the former index archives system metadata, the hierarchical index is built from domain-specific metadata extracted from the resources. An architectural overview is shown in Figure 3.

The diversity of the metadata provides us flexibility in query support, allowing us to perform and answer several interesting queries efficiently, such as the following:

1) show all the DOIs belonging to this researcher
2) show all jobs run last week by a facility user
3) show all files that contain the climate land modeling variable "total leaf area index"
4) show all files in a simulation that show increasing temperatures through an entire decade of a run
5) show all the jobs that ran in the last 6 months and are associated with any resource that has an aggregate mean of 35 degrees Celsius in the amazon forest.

It is important to note that while some of these queries are addressable by one of the three metadata catalog components (e.g. queries (1), (2), and (3)), our novel approach guarantees that other queries can be satisfied using derived metadata values (e.g. (4)) or some combination of the three components (e.g. (5)). Query 1 can be addressed with simple graph traversal. Query 2 is satisfied by the stat metadata index. Query 3 is satisfied by the resource based metadata index. Query 4 is satisfied by metadata derivation within the resource based metadata index. Query 5 requires that all three search mechanisms be used to together. We describe each of the three components below.

1) Graph Traversal: Most common queries can be resolved by examining the edges connecting resources of various types in the graph. In the previous section, we discussed basic connectivity in the context of the graph association engine, which will answer these edge traversal-based queries.

2) StatMetadata: System, or stat, metadata constitutes a major portion of data properties in an HPC facility. These properties are key pieces of information for both administrators who monitor the system behavior, as well as facility users who want to peruse their contents quickly. This information includes time information of file metadata operation (`ctime`, `mtime`, and `atime`), UID, GID, mode, and file path. Timestamp information and file path are examples of two major attributes to be searched in this manner.

The design of the stat index depends on a number of factors. The most important problem to consider is how this external index will be synchronized with the graph. As seen in Section II-A, Constellation is updated in batch mode exactly once every night using the difference engine that compares two daily snapshots of the filesystem. This procedure also provides the stat metadata we would like to capture. In the worst case, we may have to rebuild the entire search index in order to synchronize the two structures.

Figure 4 shows the basic architecture for the stat metadata search index. The architecture comprises of a search service, an HBase [21] index, and the index builder. The search service provides suitable APIs allowing the Constellation graph to pose queries against the HBase index. We chose HBase, a popular distributed key-value store, to implement the stat metadata index due to its desirable scalability characteristics (the need to scale to 750M file entries). In order to optimize the performance of the index, we employed the following techniques: (a) pre-splitting region keys to maximally parallelize the insert/update operations; (b) applying compression format for index storing in order to reduce the amount of I/O operations; and (c) enlarging the memstore size to reduce the frequency of memstore compaction, which can easily multiply I/O operations unnecessarily.

To interface with the HBase index, the search service uses Thrift [19]. The index builder constructs the inverted index from the diffs computed by the diff engine. File paths, for example, are ingested into the index by parsing the pathnames, identifying keywords (excluding common words), and using the keywords as keys and graph vertex IDs as values. The resulting key-value pairs are stored in HBase and are accessible through the search service. To this end, we have utilized Spark, due to its optimal performance for large-scale data processing. A similar approach can be adopted to index not just file stat metadata but also job script and DOI metadata as shown in the Figure 4. For example, indexing the job script metadata would allow us to query, "if a set of jobs belonged to a *campaign* of runs for the Intergovernmental panel on climate change, IPCC," where campaign is an attribute that is extracted from the PBS job script, describing the job run on Titan.

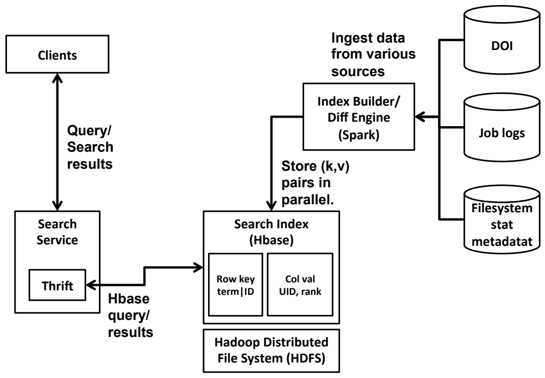
Clients
Query/ Search results
Ingest data from various sources
DOI
Index Builder/ Diff Engine (Spark)
Job logs
Store (k,v) pairs in parallel.
Search Service
Thrift
Search Index (Hbase)
Row key term|ID
Col val UID, rank
Filesystem stat metadatat
Hbase query/ results
Hadoop Distributed File System (HDFS)

Fig. 4: Architecture of stat metadata index. A client issues a search query to the search service that interfaces with HBase search index via the thrift API. The index is a KV store that ingests system metadata from Lester, which produces the daily file system snapshot.

3) Hierarchical Metadata Index: The stat index outlined above works well for queries requesting system specific information. While useful for administrators and users that know specific information about the file location or creation, the metadata captured here is not descriptive enough to support richer queries about the data itself. Query (3) in Section II-D, for example, requests to find files that contain a specific variable "total leaf area index" (a key variable in a climate land model). This field would not exist in the stat index. These types of queries are addressed by employing a second external index that contains richer fields of description. This index employs the popular high performance inverted search index Solr. Furthermore, we instituted an automated metadata derivation component that will utilize a combination of base metadata values as well as the data itself to derive new metadata. Specifically, we implemented a novel, *hierarchical* index abstraction that attempts to form a knowledge hierarchy by *deriving* new metadata from previously extracted base metadata. In this framework, we take a bottom-up approach of automatically extracting metadata from the resource fabric (e.g., scientific datasets) and collecting as much information as possible without user intervention. A similar approach can be adopted to extract metadata from other elements such as job scripts, logs, documents, charts, and tables.

The hierarchical framework is outlined in Figure 5 and consists of four distinct abstract layers. The foundational layer, layer 0, consists of metadata extracted from self describing scientific data formats (e.g., *netCDF*) that describe the experiment, simulation, authors, and variables and dimensions of the contents. Layer 1 *derives* new information about interesting aggregate values on an individual file based on the information extracted in the previous layer. These aggregate values may include a mean, a histogram, or a boolean property of the data not otherwise defined. Layer 2 defines aggregate properties over *collections* of data. Collections constitute an important aspect of scientific applications' data production behavior, and can be formed either via the data production process where sets of files (thousands) are grouped based on being generated together or based on derived layer 1 properties (e.g., files grouped based on temperature ranges). Yet another example is a user-defined collection like a DOI, where relevant resources are fused together by a common identifier. Finally, in layer

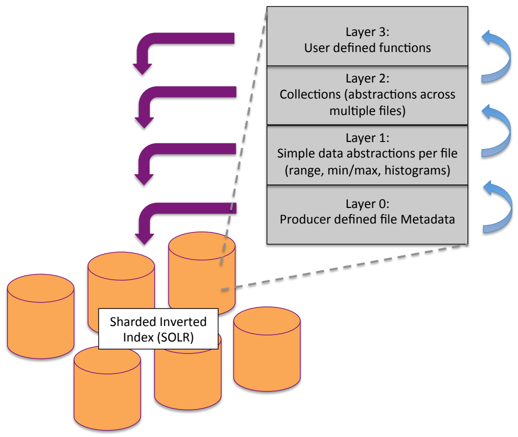
Layer 3:
User defined functions
Layer 2:
Collections (abstractions across multiple files)
Layer 1:
Simple data abstractions per file (range, min/max, histograms)
Layer 0:
Producer defined file Metadata
Sharded Inverted Index (SOLR)

Fig. 5: The multi-tiered, hierarchical search index design. At Layer 0, we extract base metadata from files and store in the index. Layer 1 utilizes basic functions to derive new metadata from the data. Layer 2 performs the same operation across collections. Layer 3 allows users to customize functions that will produce new metadata.

3 we allow publishers of data to define custom metadata properties. These customized properties are derived as output from user-defined functions over a collection of datasets.

We begin our implementation of the hierarchy by indexing metadata extracted from *netCDF* files. NetCDF is a self-describing, array oriented scientific data format, designed to allow users to inject descriptive information and seamlessly integrate it with the data through its headers and global attributes. NetCDF files contain metadata in the header, consisting of attributes and properties, and data in the body, represented as a multidimensional array (e.g. spatio-temporal dimensions). One of the main benefits of utilizing netCDF is its rich complement of tooling. One such tool, ncdump [18] extracts metadata and places it into easily parsed comma separated values. We use ncdump to grab pertinent properties, and then POST them as payload into a sharded Solr index with a pre-defined schema. The properties are posted in association with a specific nodeID in the graph and the index remains synchronized. We then send queries to Solr over distinct properties like $variable$ or $projectname$. This base information is populated into the foundational layer 0 of the hierarchy.

After layer 0 is populated, we derive metadata for the upper layers. For layer 1, we read the contents of the dataset and compute the layer 1 data. This new information will be injected into the index at higher tiers so as to further describe the data resource and provide new information for derivation. The processes apply simple, domain-agnostic functions like $mean$, min and max over individual properties of a *single* data resource. For example, for a climate dataset, we can find the minimum temperature of a specific data file within a model's time series and inject that value back into the index with the field name $mintemp$.

This background process also extracts layer 2 metadata from *collections* of data. A more complex example of oft-used collections are time series experiments used in climate models, where each file represents values of variables (e.g. atmospheric pressure, ocean salinity, etc) in a particular year. Subsequently, the files generated by the model output are grouped together for analysis over a time epoch. At this level, we apply the domain-agnostic aggregate functions to the collection, which could either be applying the function to the collection of files themselves or applying the function on the layer 1 attributes derived from the files. For example, we may compute a global minimum from the minima of individual files, or we may compute minimum temperature value over a specific grid point over a period of a 150 years and inject that value as property in the index.

The top layer (3) allows user-defined functions to be applied to individual *or* collections of datasets, so that we may derive domain-specific properties. One example a user defining an analysis function is to compute a boolean property representing whether the temperature over a specific grid point is monotonically increasing over a 10 year collection of climate model output files. Constructing layer 3 is somewhat more complicated, and is a source of future work.

The layers described in the hierarchical index will allow us to answer rich queries about datasets without the user having to provide any metadata, thereby enabling the discovery of hidden data and knowledge pathways that would have otherwise been impossible. Our current index implementation consists only of netCDF data, but we plan to extend our implementation to other file formats such as text documents, publications (in formats like word docs and pdfs), figures, charts, and other valuable resources that can leverage open source extraction tools such as Apache Tika [21].

E. Graph Analytics

In Section II-C, we described association building within the graph, as a means to derive relationships between resources. Given the rich set of resources within a large-scale collaboration there are deeper, and more meaningful connections hidden beneath the surface, which if unravelled can help answer sophisticated queries that provide insights to the end-user about different knowledge artifacts (perhaps, even across disciplines). The graph representation of the resources allows us to perform some very interesting analytics based on its structure and topology. Imagine the ability to suggest to users new resources (e.g., datasets) of interest or to administrators on how to provision resources based on either graph analytics or data mining. The graph analytics techniques primarily deal with the structural (or topological) and temporal properties of the graph to identify characteristics such as densely populated regions of the graph, popular data products or usage of resources over time, and recommend further study of such resources as well as performing actions based on them (e.g., a popular dataset may require more redundancy or identifying a specific library that is used often by codes). On the other hand, we may need data mining to go beyond the topology of the graph, to predict potential data products of interest based on a similarity in the key attributes used to produce the data.

We illustrate a specific data analytics scheme used in Constellation, namely PageRank. The graph structure of Constellation is highly similar to scale-free networks such as the World Wide Web. In particular, a few nodes act as hubs (e.g., common libraries used within the Titan system) and other nodes have sparse connections to the rest of the graph.

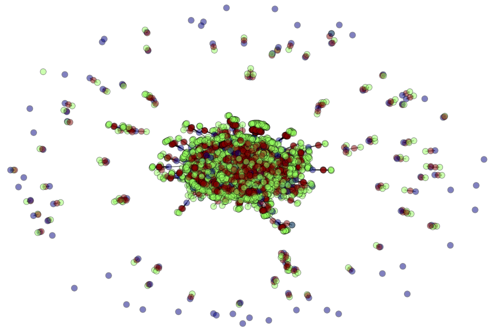

Fig. 6: The figure shows the network between users (red vertices), groups (green vertices), and projects (blue vertices). Edges are connected between users and projects as well as users and groups. We can find a large connected component with 5440 vertices out of a total of 5759 vertices, along with other 103 small connected subgraphs. The diameter of the largest connected component is 14, which means a vertex can reach any other vertex in 14 hops.

This makes it suitable to analyze the topology of the graph using PageRank. All of the files from the file system, users, groups and jobs are integrated into a single, unified graph representation. Based on the overall connectivity in the graph (see Section II-B), it is possible to use this topology to discover popular data products (vertices with many in-bound edges). Similar to web-pages that are "ranked" based on their popularity, and are used to provide customized search results to users, our implementation of PageRank provides the ability to search through the graph to quickly access relevant data.

There are specific challenges associated with the implementation of PageRank on a graph that has over 750 million entities. Similar to the Internet, the Constellation graph may not be strongly connected as shown in Figure 6, which is constructed from the information in file system snapshot data. This can lead to isolated parts of the graph having high popularity ranks despite not being well connected. To overcome this challenge, we implement a version of PageRank that removes nodes with zero out-degree and adds random jumps to avoid being limited to isolated parts of the graph. Our implementation provides near real-time performance for searching across the graph. In addition to analyzing the graph itself, it is also possible to analyze the metadata (extracted from the data) to discover related data products. For example, datasets produced using similar simulation parameters (e.g., input decks, node count) or models (e.g., climate land models) can be linked together as being potentially related. Such mining of metadata enables the discovery of new data pathways.

III. Evaluation

We implemented Constellation using a dedicated cluster of several systems (seven in total), and deployed it in the OLCF environment. The cluster consists of two-socket, quad-core machines, each containing 96 GB RAM and 1 TB of local storage. One system was used for building the Constellation graph and another was used as an external user interface for clients. One of the nodes hosted the hierarchical index, which interfaced with the metadata collection process that was run on a cluster. The remaining four were responsible for both gathering resource information from the different data sources discussed in Section II-B, as well as the Hadoop cluster required to run the diff engine. The implementation of the stat index utilized 9 virtual machines. Each VM consists of 32 virtual CPUs, 64GB of RAM, and 500GB of local storage. While we have implemented a subset of the features outlined in Figure 1, we have nevertheless found a number of encouraging results in our initial evaluation.

Graph structure evaluation: We compare the construction efficiency of the Constellation graph structure with the open source, property-based graph engine Titan:db [9] (note that this is different from the Titan machine in OLCF). Titan:db has many advantageous features, including the ability to retain node properties as well as having a well-supported query language (i.e. Gremlin). Our customized in-memory graph solution, however, compares favorably with Titan:db given the requirements of Constellation. The requirement is that large amounts of data must be inserted and updated into the graph on a daily basis, a feature that is not emphasized within the Titan:db community. Our evaluation shows that Constellation outperforms Titan:db for graph construction by 100x for a 500-million vertex insert operation. This is because Titan:db was not designed to perform graph updates in bulk and builds a synchronous index during graph construction as opposed to our decentralized index architecture.

Before the Constellation graph is updated, there are two time consuming preceding steps. The first is to launch the two data collection tools, Lester and HPSSDU. Lester, the file snapshot capture tool, takes 1.5 hours on average to complete, while HPSSDU, the HPSS snapshot capture tool, takes around 2.5 hours to complete. The subsequent step requires launching the diff engine against two consecutive snapshots. This procedure takes on average 50 $mins$ with a deviation of 20 $mins$. This deviation in time depends on the volume of data that needs to be diffed as well as the number of inserts, updates, and deletes to be computed between snapshots.

External index evaluation: We focus on the performance of the external search indexes (Section II-D). Table II demonstrates the time and space overhead of the stat metadata index, along with the response time of a few sample queries. Recall that the search index is maintained in HBase and is built using a Spark infrastructure. Building the index took 3.08 hours, including 2.14 hours for inserting the computed values into HBase. As a proof of concept, we used eight region servers for HBase and eight worker nodes for Spark. Our first test query retrieves files with the keyword "netcdf" associated with a user with UID '9486'. The query matched 17.5 million entries consuming a total of 247s for retrieval. HBase operations consumed 53s of the total time. Our second test query retrieves files with a regular expression match of "netcdf.*baseline.*" associated with user '9486'. The query matched 9.5 million entries consuming a total of 187s for retrieval, requiring 127s for HBase operations. Even though query 2 had only half as many entries in the result set as query 1, it took 75% of the time required by query 1, which may seem counterintuitive. However, query 2 involved extra overhead because of the regular expression computation. Additionally, with more hardware resources and optimization of the index, we can reduce these search times. Limiting the result set will also contribute a reduction in response time of the query. We can further overlap the retrieval of additional

TABLE II: External search indexes (average of 5 runs).

Parameter	Value
saving index to HBase	7693.89 s (2.137 hrs)
index building time (includes saving to HBase)	11109.79 s (3.08 hrs)
index space	32.2 GB
index entries	736.4 million entries
search query 1	247 s (17.5 million entries)
search query 2	187 s (9.6 million entries)

query responses as the user analyzes the initial result set.

We now focus on the evaluation of the hierarchical metadata index. According to OLCF, the usage of the Spider storage system on average is around 15PB of data (approximately 50 percent of the total capacity of 32 PB). It is desirable to maintain file system usage at a 50% capacity, as performance deteriorates when usage approaches full capacity because of the overhead in finding free storage blocks. As a result, the center periodically purges data from the file system at around 50%. This requires the user to maintain only active data (required by currently running or impending jobs) on the file system while archiving the remainder on HPSS storage. The challenge in building the hierarchical metadata index is in deriving metadata from petabytes of data as the file system snapshot is changing regularly. Note that we do not build the hierarchical metadata for the entire petascale storage system. Instead, as mentioned in Section II-D3, it was built using either a user-defined collection or a scientifically insightful dataset. The metadata extraction is performed on another cluster within the facility and launched as a parallel application on several nodes. We use the output of the diff engine described in Section II-B, which computes the difference between two consecutive file snapshots, as input to metadata extraction.

TABLE III: Time taken and index size for derived metadata.

Layer	Time (s)	Index Size (KB)
0 (per file)	< 1	6.8
1 (per file)	118	29
2 (aggregate)	38	37

In Table III, we show the time taken and volume of metadata derived for each layer of the hierarchical index (with the exception of layer 3) using the output from the Atmospheric Model Intercomparison Project (AMIP) [11] experiment within the climate domain (available on the production Earth System Grid Federation). We sampled monthly data over a decade (120 files in all) from a larger, 150-year run of the experiment. The files contain several properties (e.g. temperature, salinity) with their associated values that describe the experiment and are encoded in netCDF format. We define time in layer 0 in Table III as the time required to extract the base metadata properties and put them in the Solr index. The time measurement for layer 1 is the time needed for both computation of the basic domain-agnostic functions, like min, max, and $mean$, for individual files and the subsequent injection of the results into the index. The time in layer 2 is the time required to apply the basic functions on the derived data in layer 1 across a collections of files (i.e. the decade consisting of 120 files) and injection of aggregate values into the index. Note that the time for population of layer 1 is a cumulative measure (i.e. an aggregation of both layer 0 and layer 1). Building layer 0 data takes less than 1 second, and the index size is 6.8 KB. Layer 1 is built by reading the dataset, and consumes $118s$ for generating 29 KB of derived data for a single file. Layer 2 builds on the derived data from layer 1 for the collection, and consumes $38s$ for 37 KB of derived data. For our sample collection of 146 GB (120 files), the total time consumed was less than 4 hours and 4 MB of metadata was generated and injected into the index. Note that this process can be parallelized for faster indexing.

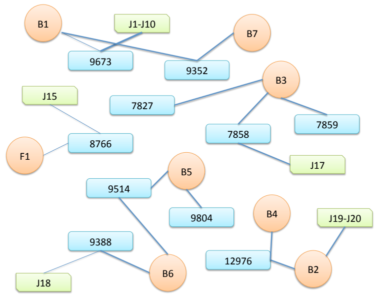

Fig. 7: Resource entities and their relationships. Blue rectangles are users, green rectangles are jobs, and orange circles are groups. The connections between users, jobs and groups shown in blue. This subset of the graph is a summary of the most popular set of resources on a particular day.

Next, we conducted a qualitative recall test comparison between our hierarchical index and the popular climate data archive Earth System Grid Federation (ESGF) [7]. Our comparison involves issuing three queries against both indexes: (Q1) Find all resources that contain the variable "Surface temperature (radiative)", (Q2) Find all resources that have an average temperature that is greater than $25C$, and (Q3) Find all collections that have an average temperature that is greater than $25C$. Table IV demonstrates that both indexes are able to satisfy queries for basic metadata lookups (as posited in Q1). More sophisticated queries over derived information, however, are not satisfiable by traditional cataloging mechanisms that our extracted knowledge index can support.

TABLE IV: Hierarchical Metadata Index vs ESGF

Query	Hierarchical Metadata Index	ESGF
Q1	yes	yes
Q2	yes	no
Q3	yes	no

Graph analytics evaluation: Finally, we evaluate some of the structural analytics algorithms discussed in Section II-E. We used a single snapshot of the graph to test our PageRank implementation. The results are intended to demonstrate the kinds of inferences one can draw from the graph about system usage, popularity of users, groups, jobs and domains.

We present a graphical representation of the data in Figure 7. Note that PageRank is run only on the basis of the graph's topology, and there is no explicit representation of 'features' such as the number of files per user or the number of jobs submitted. The ten most popular users obtained by running PageRank on the graph represent users with high utilization of the Titan supercomputer. The results also indicate that a majority of the users on that day belonged to the 'biophysics' area and the jobs run by them typically included molecular dynamics simulations. Similarly, the top user group also corresponds to the 'biophysics' area. In summary, the PageRank based graph analytics allows us to identify the popular (or dense) regions or subgraphs within Constellation as a means to study and address hotspots. Performing such an operation repeatedly over a period of time will allow us to study the temporal evolution of resource usage and user behavior, which can help with provisioning decisions, e.g., purge exemptions.

IV. Related Work

Graphs are widely adopted in enterprise systems for a wide-range of applications such as organizational and product data management, network and IT operations, real-time recommendations, fraud detection, access management and social networking [16]. Popular social networking products such as Facebook [22], LinkedIn [3] and Google Plus [15] exemplify their use. This includes the social network Research Gate [23], which is similar to our work because it applies social networking principles to the discovery of scientific publications. Our work expands this idea by representing the *entire resource fabric* of a scientific collaboration in the graph in an attempt to discover new pathways.

Several systems above utilize popular open source property graph engines such as Titan:db [9] and Neo4j [24]. Property graph is a graph representation model that describes network topology and arbitrary properties (or attributes) of graph entities. NoSQL-inspired graph databases typically support property graph model because of flexibility and expressiveness. We also evaluated a property graph for our graph representation, but chose to implement an in-memory graph due to the query performance and bursty bulk loading of the graph.

Research in the optimization of metadata search and discovery in large scale environments has been gaining attention in HPC. Leung et al. [14] address the difficulty of managing millions of files for large scale storage systems with *Spyglass*, which improves the search process by pruning the search space. They accomplish this by using a novel index partitioning mechanism that leverages namespace locality combined with a signature file format to describe a partition's contents compactly. This work, however, only captures system metadata, such as filename or timestamp, and ignores the wealth of metadata about the content available within the file itself. Similarly, Hua et al. [13] and Wei et al. [25] optimize the search process by adopting Bloom filters that further prune the search space. Dai et al [8] use Darshan, an MPI library that can be linked to applications to trace user, job, and file I/O information and link them up using property graph. Our work is fundamentally different from Darshan's client-based strategy in that we adopt a bottom up, non-intrusive systems approach to collect information from the resource fabric. Propeller [27] utilizes access-causality graphs (ACGs) that capture file-access patterns. Propeller's realtime indexing scheme is designed based on the observation that file accesses of analytics applications tend to frequently cluster amongst and around correlated files. While this intuition led to better file-search performance, its focus was only on stat metadata. Popular climate data archives such as Earth System Grid Federation [7] and ARM [26] attempt to catalog content based metadata on a large-scale within the climate domain using controlled vocabularies. These implementations provide richer metadata searches for both facility users as well as external clients that are interested in the data. We expand their capabilities by deriving new metadata from previous sources.

V. Acknowledgments

This research used resources of OLCF, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] Pegasus: Mapping scientific workflows onto the grid. In *Grid Computing*. Springer, 2004.

[2] Apache Solr. http://lucene.apache.org/solr/.

[3] A. Auradkar, C. Botev, et al. Data infrastructure at linkedin. In *IEEE International Conference on Data Engineering (ICDE)*, 2012.

[4] A. S. Bland, J. C. Wells, O. E. Messer, O. R. Hernandez, and J. H. Rogers. Titan: Early experience with the Cray XK6 at Oak Ridge National Laboratory. In *Cray User Group Conference (CUG)*, 2012.

[5] C. Chang, S. Ku, and H. Weitzner. Numerical study of neoclassical plasma pedestal in a tokamak geometry. *Physics of Plasmas*, 11(5):2649–2667, 2004.

[6] J. Chen, W. Geyer, C. Dugan, M. Muller, and I. Guy. Make new friends, but keep the old: recommending people on social networking sites. In *ACM Conference on Human Factors in Computing Systems*, 2009.

[7] L. Cinquini, D. Crichton, et al. The earth system grid federation: An open infrastructure for access to distributed geospatial data. *Future Generation Computer Systems*, 36, 2014.

[8] D. Dai, R. B. Ross, P. Carns, D. Kimpe, and Y. Chen. Using property graphs for rich metadata management in hpc systems. In *Parallel Data Storage Workshop (PDSW), 2014 9th*, pages 7–12. IEEE, 2014.

[9] T. distributed graph database. http://thinkaurelius.github.io/titan/.

[10] J. Dongarra, H. Meuer, and E. Strohmaier. Top500 supercomputing sites. http://www.top500.org, 2009.

[11] W. L. Gates. Amip: The atmospheric model intercomparison project. *Bulletin of the American Meteorological Society*, 73(12), 1992.

[12] R. L. Henderson. Job scheduling under the portable batch system. In *Job scheduling strategies for parallel processing*. Springer, 1995.

[13] Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian. Supporting scalable and adaptive metadata management in ultralarge-scale file systems. *Parallel and Distributed Systems, IEEE Transactions on*, 22(4):580–593, 2011.

[14] A. W. Leung, M. Shao, T. Bisson, S. Pasupathy, and E. L. Miller. Spyglass: Fast, scalable metadata search for large-scale storage systems. In *FAST*, volume 9, pages 153–166, 2009.

[15] G. Magno, G. Comarela, D. Saez-Trumper, M. Cha, and V. Almeida. New kid on the block: Exploring the google+ social graph. In *ACM conference on Internet measurement conference*, 2012.

[16] M. Newman, D. Watts, and S. Strogatz. Random graph models of social networks. *Proceedings of the National Academy of Sciences*, 99, 2002.

[17] M. Papka, S. Coghlan, E. Isaacs, M. Peters, and P. Messina. Mira: Argonne's 10-petaflops supercomputer. Technical report, ANL (Argonne National Laboratory (ANL), Argonne, IL (United States)), 2013.

[18] R. Rew and G. Davis. Netcdf: an interface for scientific data access. *Computer Graphics and Applications, IEEE*, 10(4):76–82, 1990.

[19] M. Slee, A. Agarwal, and M. Kwiatkowski. Thrift: Scalable cross-language services implementation. *Facebook White Paper*, 2007.

[20] G. Staples. Torque resource manager. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 8. ACM, 2006.

[21] The Apache Software Foundation. http://www.apache.org/.

[22] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the facebook social graph. *arXiv preprint arXiv:1111.4503*, 2011.

[23] R. Van Noorden. Online collaboration: Scientists and the social network. *Nature*, 512(7513):126–129, 2014.

[24] J. Webber. A programmatic introduction to neo4j. In *ACM Systems, Programming, and Applications: Software for Humanity*, 2012.

[25] J. Wei, H. Jiang, K. Zhou, and D. Feng. Efficiently representing membershipfor variable large data sets. *Parallel and Distributed Systems, IEEE Transactions on*, 25(4):960–970, 2014.

[26] S. Xie, R. B. McCoy, et al. Clouds and more: Arm climate modeling best estimate data: A new data product for climate studies. *Bulletin of the American Meteorological Society*, 91(1), 2010.

[27] L. Xu, H. Jiang, L. Tian, and Z. Huang. Propeller: A scalable real-time file-search service in distributed systems. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2014.